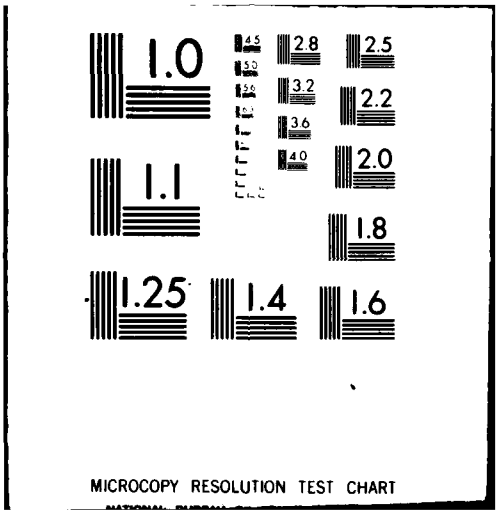
AD-A084 282 TEXAS UNIV AT AUSTIN CENTER FOR CYBERNETIC STUDIES F/6 12/2
A PRIMAL SIMPLEX VARIANT FOR THE MAXIMUM FLOW PROBLEM,(U)
DEC 79 F GLOVER, D KLINGMAN, J MOTE N00014-78-C-0222
UNCLASSIFIED CCS-362 NL

END
DATE FILMED
6-80
DTIC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART

(6) LEVEL II

CENTER FOR CYBERNETIC STUDIES

The University of Texas
Austin, Texas 78712

(14)
CCS-362

(6)
A PRIMAL SIMPLEX VARIANT FOR THE MAXIMUM FLOW PROBLEM

by

(10)
Fred Glover*
Darwin Klingman**
John Mote***
David Whitman****

(12) 46

(11)
December 1979

* Professor of Management Science, University of Colorado, Boulder, CO 80309

** Professor of Operations Research and Computer Sciences, BEB 608, The University of Texas at Austin, Austin, TX 78712

*** Systems Analyst, Analysis, Research, and Computation, Inc., P.O. Box 4067, Austin, TX 78765

**** Systems Analyst, Analysis, Research, and Computation, Inc., P.O. Box 4067, Austin, TX 78765

(15)
This research was partially funded by the U.S. Department of Transportation, Contract No. DOT-OS-70074 and by the Office of Naval Research, Contract No. N00014-78-C-0222. Reproduction in whole or in part is permitted for any purpose of the United States Government.

CENTER FOR CYBERNETIC STUDIES

A. Charnes, Director
Business-Economics Building, 203E
The University of Texas
Austin, TX 78712
(512) 471-1821

ABSTRACT

This paper presents a number of specialized implementations of the primal simplex algorithm for the maximum flow problem. Computational results indicate that some of these variants are both faster and require less computer storage than our best implementation of the classic labeling algorithm for maximum flow problems.

1.0 INTRODUCTION

For a number of years the maximum flow network problem has attracted the attention of prominent researchers in network optimization. Since the ground-breaking work of Ford and Fulkerson [9, 15, 16, 17, 18], a variety of algorithms [1, 5, 6, 11, 12, 13, 14, 26, 27, 29, 30, 32, 33, 37] featuring good "worst-case" bounds have been proposed for this problem. Surprisingly though, there has been almost no empirical evaluations of these algorithms.

Cheung [6] recently conducted the first significant computational investigation of maximum flow methods, testing several of the major approaches. Although an important step in the right direction, Cheung's implementations employ methodology and data structures originating at least a dozen years ago [5].

In the past decade, however, advances in network implementation technology have been dramatic. Improved labeling techniques and more effective data structures have (a) decreased total solution time and/or (b) reduced computer memory requirements [2, 4, 10, 19, 20, 21, 24, 25, 35, 36]. As a result, widely held beliefs about which algorithms are best for particular problem classes have been steadily challenged and in some cases completely overturned [10, 19, 21, 28, 35, 36]. This study likewise discloses several misconceptions about maximum flow algorithms whose challenge was overdue.

The emergence in recent years of the primal simplex algorithm as a superior solution technique for the more general classes of network

flow problems prompted us to develop a number of specialized implementations of the primal simplex algorithm for the maximum flow problem. Somewhat to our surprise, these primal codes can solve problems both faster and with less computer storage than our best implementation of the classic labeling algorithm for maximum flow problems.

2.0 PROBLEM DEFINITION

Let G(N,A) be a directed network consisting of a finite set N of nodes and a finite set A of arcs, where each arc $k \in A$ may be denoted as an ordered pair (u,v) (referring to the fact that the arc is directed from node $u \in N$ to node $v \in N$). Associated with each arc $k = (u,v)$ is a flow variable x_k and an upper bound or capacity coefficient u_k. Additionally, two specific nodes $s \in N$ and $t \in N$ are called the source and terminal, respectively.

The standard formulation of the maximum flow problem from a source node (s) to a terminal node (t) is given by:

$$\text{Maximize} \quad x_o \tag{1}$$

subject to:

$$-\sum_{k \in FS(s)} x_k + \sum_{k \in RS(s)} x_k = -x_o \tag{2}$$

$$-\sum_{k \in FS(t)} x_k + \sum_{k \in RS(t)} x_k = x_o \tag{3}$$

$$-\sum_{k \in FS(i)} x_k + \sum_{k \in RS(i)} x_k = 0, \quad i \in N - \{s,t\} \tag{4}$$

$$0 \leq x_k \leq u_k, \text{ for all } k \in A \qquad (5)$$

$$x_o \geq 0 \qquad (6)$$

where $FS(i) = \{k:k = (i,j) \in A\}$ and $RS(i) = \{k:k = (j,i) \in A\}$. FS(i) is called the *forward star* of node i and is the subset of arcs that originate at node i. Correspondingly, RS(i) is called the *reverse star* of node i and is the subset of arcs that terminate at node i. Where there is no danger of confusion, we will also sometimes refer to the nodes that are endpoints of a (forward or reverse) star as elements of the star itself. It is standardly assumed that RS(s) and FS(t) are empty, and hence sums of flows over arcs in these sets are often not included in (2) and (3).

3.0 EXPERIMENTAL DESIGN

3.1 Overview

Alternative implementation methods are evaluated in this study by solving a diverse set of randomly generated maximum network flow problems using the same computer (a CDC 6600) and the same FORTRAN compiler (MNF). All codes were executed during periods of comparable demand for computer use and were implemented by the same systems analysts with no attempt to exploit machine hardware characteristics.

Even with these safeguards, minor differences between the solution times of any two codes, for a single test run of each, are of questionable significance. For this reason, most of the problem networks were solved five times (i.e., for five different source-terminal pairs) and the median solution time reported. Each code makes use of a real-time

clock routine supplied by the computer vendors. Such routines can be accessed by a FORTRAN subroutine call and are generally accurate to two decimal places.

Problem data are input ("read in") to each code in exactly the same fashion (arc by arc, in the same "random" order). Since we implemented different data structures in the various codes it is necessary to consider two timing statistics when comparing codes. The *total solution time* records the elapsed time after input of the network and prior to the output of its solution. This includes the time required to initialize the function arrays. However, codes that are able to store the networks as originally input have an obvious advantage under this timing procedure. Consequently, a second solution statistic for each problem measures only the problem *optimization time* and disregards the time required to arrange the problem data in the function arrays and to retrieve the solution in a suitable output form.

3.2 Test Problems

The primary objective of this research effort was to design a solution algorithm that can be used to solve the large scale maximum flow problems that arise in the design and analysis of real-world transportation systems. In order to evaluate the solution capabilities of the numerous algorithmic variants and refinements that were studied, a data base of test problems was required. The design of this data base was the focus of a considerable amount of effort and a number of researchers in the network area were contacted regarding their opinions on problem structures most appropriate for investigation.

As a result of this analysis, four different classes of test problems were selected for the data base. Many members of each class of problems, with varying numbers of nodes and arcs, and varying arc capacities, were generated in order to analyze the effects of the problem dimensions on algorithmic solution capabilities. Altogether, over one hundred and fifty test problems were included in the data base.

Each of the four major classes of test problems will be described briefly. A uniform probability distribution was used in all instances to randomly select items such as nodes and upper bounds.

The first and simplest class of test problems consists of unstructured or "random" networks. Such a network is constructed by initially identifying the node set N (whose elements may be assumed to be numbered from one to $|N|$). The set of arcs, A, is generated by successively selecting ordered pairs of nodes, $u \in N$ and $v \in N - \{u\}$ thus creating an arc (u,v) for each pair selected. Multiple arcs directed from node u to node v are not allowed in this, or the other three, classes of test problems. The integer upper bounds, or arc capacities, are selected within a pre-defined interval. The source node, s, and the terminal node, t, are randomly chosen from the elements of N.

Due to the simplicity of the arc generation process, this class of problems possesses no specific underlying structure, and hence is referred to as the *random* class. Twelve different sets of problem dimensions, R1, R2,..., R12, were selected for this class, each containing five different problems. The specific problem dimensions are provided in Table I. Random problems were included in the study because

TABLE I

RANDOM PROBLEMS

PROBLEM	|N|	|A|	ARC CAPACITY RANGE
R1	250	1250	1-100
R2	250	1875	1-100
R3	250	2500	1-100
R4	500	2500	1-100
R5	500	3750	1-100
R6	500	5000	1-100
R7	750	3750	1-100
R8	750	5825	1-100
R9	750	7500	1-100
R10	1000	5000	1-100
R11	1000	7500	1-100
R12	1000	10000	1-100

they represent the closest analogy to test problems used by Cheung [6] and because most maximum flow literature makes no reference to any particular problem structure.

The second class of problems is called the *multi-terminal random* class. Unlike a random network, a multi-terminal random network possesses a small degree of underlying structure. The source and terminal nodes for these problems actually play the role of master source and master terminal. This results by assigning infinite capacities to all arcs incident upon these two nodes, so that all nodes in the forward star of the source

node serve as "effective sources," and all nodes in the reverse star of the terminal node serve as "effective terminals."

The effect of this construction is to create a problem that simulates a true multiple source and multiple terminal network. While the objective of a random network problem is to determine the maximum flow from the source node s to the terminal node t, the objective of a multi-terminal random network problem is to determine the maximum flow from the set of effective source nodes $\{v|(s,v) \in A\}$ to the set of effective terminal nodes $\{u|(u,t) \in A\}$. It is important to distinguish between "true" and "simulated" problems because prior knowledge that a problem contains multiple sources and terminals can be used to re-design an algorithm to make it more effective for this situation. One goal of this study was to pose maximum flow problems of alternative structures without "informing" the algorithm in advance what those structures were.

Twelve different sets of problems, MR1, MR2,..., MR12, were selected for the multi-terminal random class, and five problems were generated for each set. All problems from a given set share the same problem dimensions. Specific problem dimensions are indicated in Table II.

The third class of problems introduces additional structure into the network. This problem class is called the *transit grid* class. The source and terminal nodes again serve as master source and master terminal, as in the multi-terminal random problems, implicitly creating a set of effective sources and effective terminals. All nodes other than s and t are referred to as grid nodes, which can be viewed as arranged in a rectangular grid of r rows and c columns. Every adjacent pair of grid nodes is con-

TABLE II

MULTI-TERMINAL RANDOM PROBLEMS

PROBLEM	|N|*	|A|	AVERAGE NO. OF ARCS INCIDENT ON EACH MASTER SOURCE (TERMINAL)	ARC CAPACITY RANGE**
MR1	250	1250	5.0	1-100
MR2	250	1875	7.5	1-100
MR3	250	2500	10.0	1-100
MR4	500	2500	5.0	1-100
MR5	500	3750	7.5	1-100
MR6	500	5000	10.0	1-100
MR7	750	3750	5.0	1-100
MR8	750	5825	7.5	1-100
MR9	750	7500	10.0	1-100
MR10	1000	5000	5.0	1-100
MR11	1000	7500	7.5	1-100
MR12	1000	10000	10.0	1-100

*There were five master source nodes and five master terminal nodes.
**Excluding arcs entering or leaving source and terminal nodes.

nected by two oppositely directed arcs whose capacities are selected from a pre-defined interval.

Like the multi-terminal random class, this class of problems simulates multiple source and multiple terminal networks (and the algorithms are not amended to capitalize on this fact). Unlike the random and the multi-terminal random networks, the additional structure of the transit grid networks closely resembles that arising in urban transit planning networks. In this

setting, the grid structure captures the form of transportation routes in the greater suburban area. Source nodes represent major transit exchanges or vehicle storage facilities and terminal nodes correspond to collection nodes which are connected to key demand points within the city.

Eight different sets of problem dimensions were chosen for the transit grid class. Again, five different problems were generated for each set of dimensions. These sets of transit grid problems are referred to as TG1, TG2,..., TG8. Table III contains the specific problem dimensions for these sets.

TABLE III

TRANSIT GRID PROBLEMS

PROBLEM	|N|*	|A|	AVERAGE NO. OF ARCS INCIDENT TO EACH MASTER SOURCE (TERMINAL)	ARC CAPACITY RANGE**
TG1	235	1240	40	1-100
TG2	235	1640	80	1-100
TG3	410	2120	60	1-100
TG4	410	2720	120	1-100
TG5	635	3200	80	1-100
TG6	635	4000	160	1-100
TG7	910	4480	100	1-100
TG8	910	5480	200	1-100

*Including five master source nodes and five master terminal nodes.

**Excluding arcs entering or leaving master source and master terminal nodes.

The final class of test problems possesses the most elaborate structure. This class consists of totally dense, acyclic networks involving an even number of nodes. Every pair of nodes is connected by an arc directed from the node with the smaller node number to the node with the larger node number. The capacity of the arc (u,v) is 1 if $v > u + 1$ and is $1 + (u - \frac{|N|}{2})^2$ if $v = u + 1$. Node 1 is the source node and node N is the terminal node.

Although somewhat artificial, this class of problems was included because it was expected to require a large number of iterations (starting from a zero flow initial state) since the optimal solution is obtained when the flow on every arc in the network is at its upper bound. This class is referred to as the *hard* class. Five problems, H1, H2,..., H5, were considered differentiated by their dimensions. Table IV presents the relevant parameters.

TABLE IV

HARD PROBLEMS

PROBLEM	|N|	|A|	ARC CAPACITY RANGE
H1	20	190	1-82
H2	40	780	1-362
H3	60	1770	1-782
H4	80	3160	1-1522
H5	100	4950	1-2402

4.0 PRIMAL SIMPLEX VARIANT

4.1 Algorithm and Implementation Overview

Our specialization and alternative implementations of the primal simplex method for the maximum flow network problem begin with an idea which has also independently been proposed by J. Shapiro [34], and more recently observed again by Goldfarb and Grigoriadis [23]. Specifically, we rewrite problem (1)-(6) in an equivalent form that makes it possible to isolate a particular basis structure. The equivalent formulation is:

Maximize y_1 (7)

subject to:

$$-\sum_{k\in FS(s)} x_k + \sum_{k\in RS(s)} x_k + y_1 = 0 \quad (8)$$

$$-\sum_{k\in FS(t)} x_k + \sum_{k\in RS(t)} x_k - y_2 = 0 \quad (9)$$

$$-\sum_{k\in FS(i)} x_k + \sum_{k\in RS(i)} x_k = 0 \quad i\in N-\{s,t\} \quad (10)$$

$$-y_1 + y_2 = 0 \quad (11)$$

$$0 \leq x_k \leq u_k \quad k \in A \quad (12)$$

$$0 \leq y_1, y_2 \quad (13)$$

The preceding formulation arises by augmenting the original network $G(N,A)$ by an additional node associated with equation (11) and two additional arcs associated with the variables y_1 and y_2. Letting d denote the additional node and $G(\bar{N},\bar{A})$ denote the full associated network, then $\bar{N} = N \cup \{d\}$ and $\bar{A} = A \cup \{(t,d), (d,s)\}$.

Problem (7)-(13) constitutes a special circulation format for problem (1)-(6). Obviously, problem (1)-(6) could have been circularized more compactly by simply moving the right hand side of (2) and (3) to the left hand side, thereby implicitly adding an arc from the terminal to the source. The reason for using instead the format of (7)-(13) will soon become apparent.

In this variant of the primal simplex method, the basis tree $T(\bar{N},\bar{A}_T)$ is distinguished by the choice of node d as the root. Furthermore, without loss of generality, we may assume that arcs (d,s) and (t,d) are basic and are thus in $\bar{A}_T$. Consequently, nodes s and t always hang from the root d. This special organization enables the remaining nodes $\bar{N}$ - {d,s,t} to be partitioned into two subsets: those hanging below node s and those hanging below node t. For ease of discussion we refer to nodes on the *s-side* and *t-side* of the tree. (This node partitioning corresponds to a cut in graph terminology.) A small example will be used to illustrate the structure of the basis tree for the maximum flow problem. A 10-node and 21-arc network is given in Figure 1. The bounds on the allowable flows are specified in parenthesis beside the associated arcs. An example of a basis tree for this problem is given in Figure 2.

Building on this foundation, we undertook to test two ways of storing the original problem data as an initial step to implementing our specialized primal simplex maximum flow algorithm. The first storage scheme, called the *random* form, uses three $|A|$ length lists, FROM, TO, and CAP, to store the original problem data without arranging

FIGURE 1

EXAMPLE NETWORK

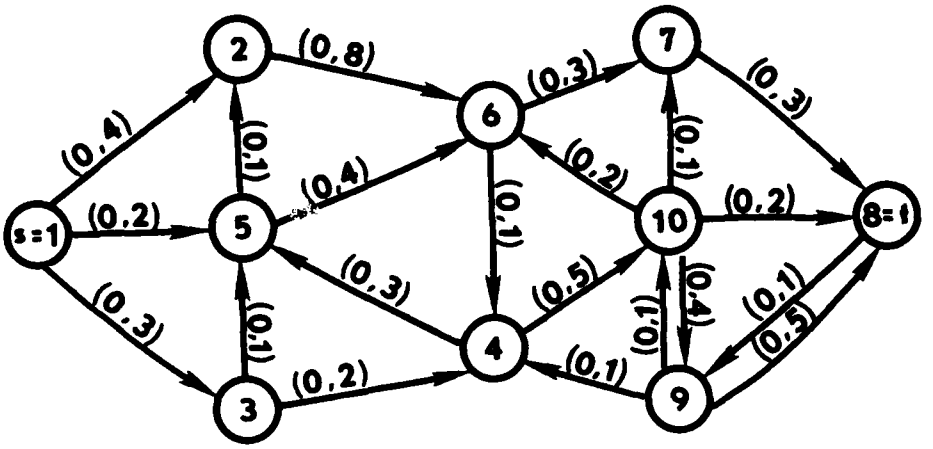
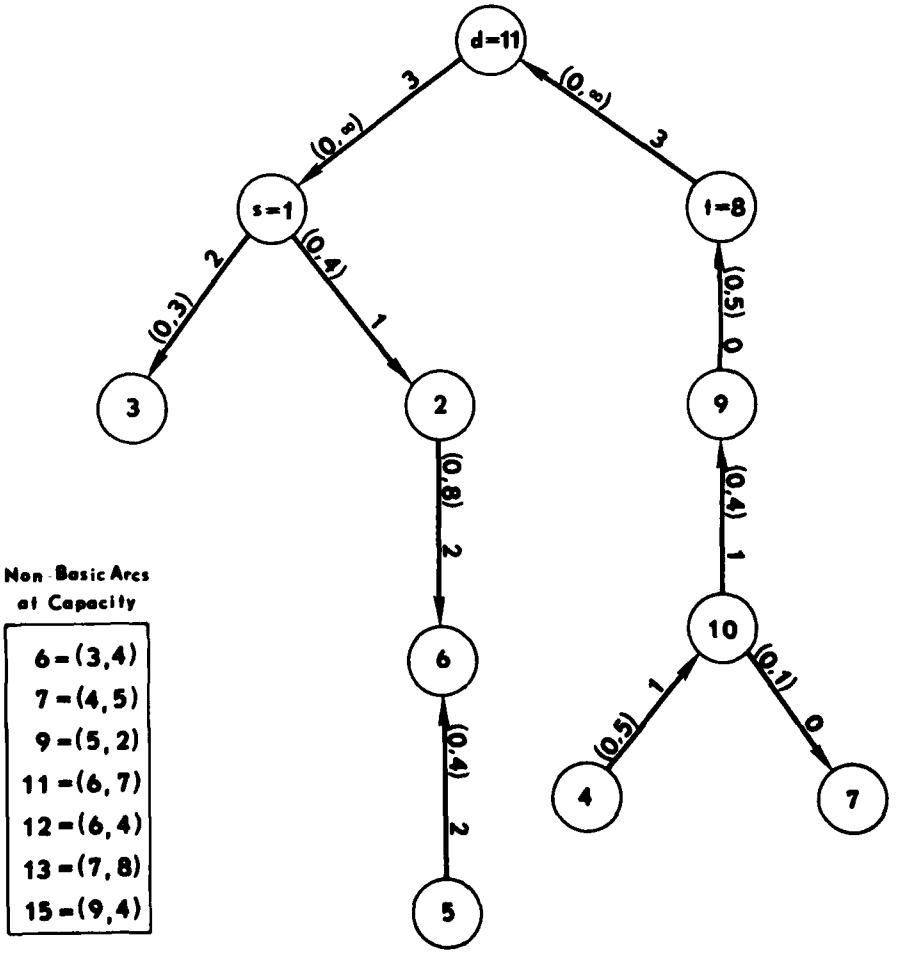


FIGURE 2

FEASIBLE BASIS TREE



Non Basic Arcs at Capacity

6=(3,4)
7=(4,5)
9=(5,2)
11=(6,7)
12=(6,4)
13=(7,8)
15=(9,4)

them in any particular order. This effectively restricts the algorithm to simple sequential processing of the arcs, since the data structure does not allow individual forward or reverse stars to be accessed efficiently. The random form data structure is illustrated in Figure 3 for the small problem given in Figure 1.

The second data structure used in the testing of the primal maximum flow algorithm is called the *forward star* form. With this data structure the arcs must be sorted according to common origin nodes, storing arcs of a given forward star in contiguous locations in the arc data lists. The $|A|$ length list FROM is replaced by a $|N|$ length list, OUT, that points to the location in the arc data lists of the first arc in each forward star. In addition to this $|N|$ length list, the forward star form uses two $|A|$ length lists, TO and CAP. The forward star form data structure is illustrated in Figure 4.

To implement these data representation schemes in a truly effective manner, we use six $|N|$ length lists to store and update the basis tree and the corresponding primal and dual solutions. These lists are: the predecessor node (PN), predecessor arc (PA), thread (THREAD), depth (DEPTH), node potential (POT), and net capacity (NETCAP). Descriptions of recent effective uses of these lists in other network contexts can be found, for example, in [2, 3, 4].

The predecessor node is an "upward pointer" in the basis tree. It identifies the unique node directly above each node in the basis tree (except the root). The predecessor arc is the arc number of the basic arc connecting a node to its predecessor node and provides access to

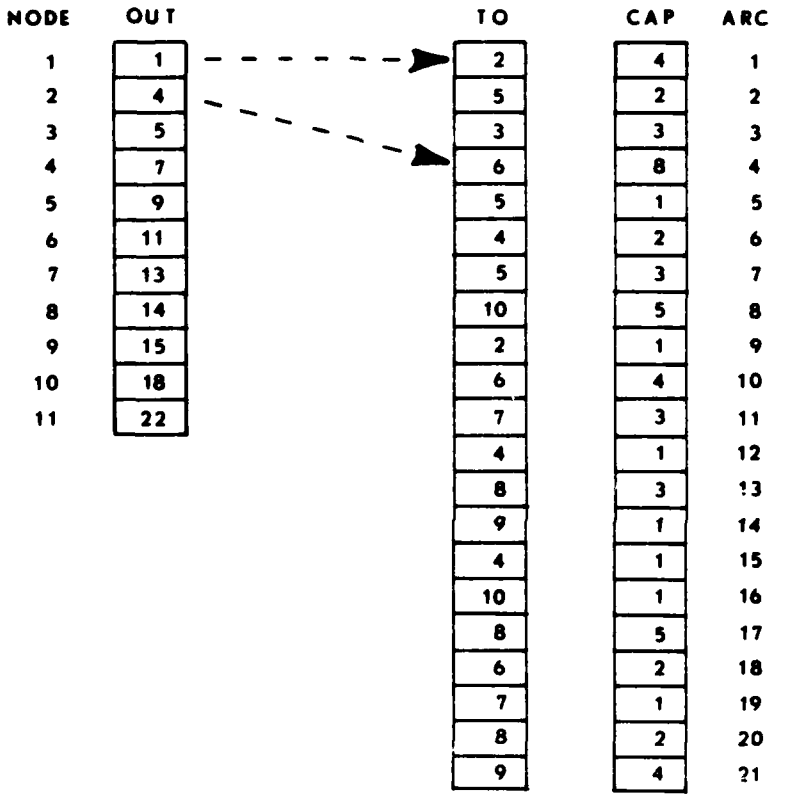
FIGURE 3

RANDOM FORM

ARC	FROM	TO	CAP
1	1	2	4
2	2	6	8
3	5	2	1
4	5	6	4
5	6	7	3
6	6	4	1
7	10	6	2
8	7	8	3
9	10	7	1
10	10	8	2
11	9	8	5
12	8	9	1
13	9	10	1
14	10	9	4
15	9	4	1
16	4	10	5
17	3	4	2
18	4	5	3
19	1	5	2
20	1	3	3
21	3	5	1

FIGURE 4

FORWARD STAR FORM



the original problem data. The thread, by contrast, is a circular list that links the nodes of the basis tree in a top to bottom, left to right fashion, enabling all nodes in any subtree to be traced in unbroken succession. The depth of the node is simply the number of arcs on the unique path in the basis tree from the node to the root, while the node potentials are the dual variables associated with the current basis tree. Finally, the net capacity is the amount of allowable flow change on the predecessor arc in the direction from a node to its predecessor node. That is, if the predecessor arc is pointed down (up) in the basis tree, then the net capacity of the arc is simply the current flow on the arc (upper bound minus current flow). This unorthodox way of storing relevant solution data for the problem actually simplifies the solution procedure. Figure 5 gives the $|N|$ length lists associated with the basis tree in Figure 2.

By associating a dual variable π_i with each node $i \in \bar{N}$, the complementary slackness property of linear programming implies the following:

$$-\pi_d + \pi_s = 1 \tag{14}$$

$$\pi_d - \pi_t = 0 \tag{15}$$

$$-\pi_i + \pi_j = 0 \quad \text{for all } (i,j) \in \bar{A}_T - \{(t,d), (d,s)\} \tag{16}$$

Since redundancy allows the value of any one of the π_i to be set arbitrarily, we elect to set $\pi_d = 0$. This choice, made natural by the choice of d as root, yields a solution to (14)-(16) such that $\pi_i = 1$ if node i is on the s-side of the basic tree and $\pi_i = 0$ if i is on the t-side of the basis tree. This property, also noted in [34], is illustrated in Figure 5.

FIGURE 5

NODE LENGTH LISTS ASSOCIATED WITH FIGURE 2

NODE	PN	PA	THREAD	DEPTH	POT	NETCAP
1	11	23	3	1	1	3
2	1	1	6	2	1	1
3	1	3	2	2	1	2
4	10	8	7	4	0	4
5	6	10	8	4	1	2
6	2	4	5	3	1	2
7	10	19	11	4	0	0
8	11	22	9	1	0	$\infty - 3$
9	8	17	10	2	0	5
10	9	21	4	3	0	3
11			1	0	0	

It may be remarked that the node potentials associated with an optimal basis tree also provide valuable information regarding the minimum cut problem. Specifically, the dual variables indicate which side of the cut the node lies on; i.e., node i is on the s-side of the cut if and only if $\pi_i = 1$. Arcs whose capacities define the capacity of the cut are also identified by the node potentials: the capacity of arc (i,j) is included in the capacity of the cut if $\pi_i \neq \pi_j$. Additionally, the arc is directed from s (t) to t (s) if $\pi_i = 1$ (0) and $\pi_j = 0$ (1).

In view of the foregoing observations, the general form of the steps of this specialized primal simplex algorithm for the maximum flow problem may be summarized as follows:

STEP 1: [INITIALIZATION] Select an initial feasible basis tree rooted at node d.

STEP 2: [DUAL SOLUTION] Determine the potential π_i of each node, according to the preceding observations, that results by setting $\pi_d = 0$.

STEP 3: [ENTERING ARC] Select a non-basic arc e from node i to node j such that (a) $\pi_i = 1$, $\pi_j = 0$, and $x_e = 0$, or

(b) $\pi_i = 0$, $\pi_j = 1$, and $x_e = u_e$.

If no such arc exists, stop. The optimal solution has been found.

STEP 4: [LEAVING ARC] If $x_e = 0$ ($x_e = u_e$), determine the maximum amount, δ, that flow can be increased (decreased) on arc (i,j) by changing the flows on the unique path from node i

to node j in the current basis tree. If $\delta \geq u_e$, go to STEP 6.

STEP 5: [CHANGE OF BASIS] If $x_e = u_e$, set $\delta = -\delta$. Let arc r be one of the arcs that only allows a flow change of $|\delta|$ in STEP 4. Change the flow by δ on the unique path from node i to node j in the basis tree. Change the flow on arc e by δ. Replace arc r with arc e in the set of basic arcs A_T and update the basis tree labels. Go to STEP 3.

STEP 6: [NO CHANGE OF BASIS] If $x_e = u_e$, set $\delta = -u_e$. Otherwise set $\delta = u_e$. Change the flow by δ on the unique path from node i to node j in the basis tree. Change the flow on arc e by δ. Go to STEP 3.

The specific implementation of each step of the preceding algorithm, of course, plays a major role in determining the overall efficiency of a particular computer code. During the extensive testing of this primal simplex maximum flow algorithm, over twenty alternative implementations of the basic algorithm were developed. These alternatives were used to determine the impact of the choice of the starting basis (STEP 1), selection of the entering arc (STEP 3), and the selection of the leaving arc (STEP 4).

Techniques whose efficiency has been well-established were used to update the list structures (STEPS 2, 5, and 6). We exploited the fact that the unique basis equivalent path (for any entering arc) contains the root node d, thereby making it possible to update the flows on the arcs in this path in a single pass. This one pass update is efficiently carried out by using the predecessor node and depth

functions (in conjunction with the net capacity function).

The next subsections contain descriptions of the various start and pivot strategies examined. The final subsection contains the results of our testing.

4.2 Starts

Four basic implementations of STEP 1 were tested during the course of this study. The first implementation is referred to as the *LIFO label-out start* procedure. The basic steps of this procedure are:

STEP 1A: [INITIALIZATION] Initialize the predecessor node function:

$$PN(d) = 0,\ PN(s) = d,\ PN(t) = d.$$

Initialize a node label function:

$$LABEL(d) = -1$$

$$LABEL(s) = -1$$

$$LABEL(t) = -1$$

$$LABEL(i) = 0 \text{ for all } i \in N - \{s,t\}$$

Set i = s.

STEP 1B: [NODE SCAN] For each arc (i,j) in the forward star of node i, check the label status of node j. If LABEL(j) = 0, then set PN(j) = i, LABEL(j) = LABEL(i) and LABEL(i) = j.

STEP 1C: [NEXT NODE] Set i = LABEL(i). If i > 0 go to STEP 1B. Otherwise, the construction of the initial LIFO label-out tree is complete.

This implementation uses the node label function (LABEL) as a depth first sequence list. That is, each newly labeled node is placed at the

front of the sequence list. Since this approach only requires a single pass through the arc data, it uses very little c.p.u. time to execute. The creation of the necessary node function values are easily incorporated into STEP 1B so that all initialization is carried out during the construction of the initial basis tree.

The initial solution constructed by the LIFO label-out start procedure exhibits the following characteristics: all arcs, basic as well as non-basic, have zero flow; all nodes except the terminal and the root are on the s-side of the basis tree; and all basic arcs, except (t,d) are directed away from the root.

The second implementation of STEP 1 is called the *FIFO label-out start* procedure. This start is identical to the LIFO label-out start except that each newly labeled node is placed at the back, instead of the front, of the sequence (LABEL) list. Thus it has the characteristics indicated for the LIFO procedure.

The primary difference between the two methods is the shape of the basis tree. FIFO, the breadth first start, tends to construct a wide, shallow initial tree whereas LIFO, the depth first start, tends to construct a narrow, deep initial tree.

The third implementation of STEP 1 that was tested is called the *balanced tree start* procedure. Unlike the other implementations, this procedure requires multiple passes of the arc data, and therefore more c.p.u. time is required to execute the start. However, this start procedure, as its name implies, attempts to balance the number of the nodes on the s-side and t-side of the basis tree, and was motivated by

the fact that the only pivot eligible arcs (STEP 3) are those whose nodes are on opposite sides of the basis tree. By balancing the number of nodes on each side, we expected that the entering arc could be selected more efficiently. The steps of the balanced tree start are outlined below.

STEP 1A: [INITIALIZATION] Initialize the predecessor node function:

$$PN(d) = 0,\ PN(s) = d,\ PN(t) = d.$$

Initiate a node label function:

$LABEL(d) = -1$

$LABEL(s) = -1$

$LABEL(t) = 1$

$LABEL(i) = 0$ for all $i \in N - \{s,t\}$

Set $i = s$.

STEP 1B: [NODE SCAN CHOICE] If $LABEL(i) = -1$, go to STEP 1D. If $LABEL(i) = 0$, go to STEP 1E.

STEP 1C: [NEXT NODE] If a complete pass of the arc data fails to relabel any nodes, go to STEP 1F. Otherwise, select a node i. Go to STEP 1B.

STEP 1D: [SCAN DOWN] Set $LABEL(i) = -2$. For each arc (i,j) in the forward star of node i, check the label status of node j. If $LABEL(j) = 0$, then set $PN(j) = i$ and $LABEL(j) = -1$. Go to STEP 1C.

STEP 1E: [SCAN UP] Select an arc (i,j) in the forward star of node i such that $LABEL(j) = 1$. If no such arc exists, go to STEP 1C. Otherwise let $PN(i) = j$ and $LABEL(i) = 1$. Go to STEP 1C.

STEP 1F: [FINAL PASS] For each arc (i,j) such that LABEL(i) = 0 and LABEL(j) $\neq$ 0 let PN(i) = j and LABEL (i) = 1.

The initial multiple passes through the arc data (STEPS 1C, 1D, and 1E) attempt to select as many arcs as possible that are either directed away from the root node on the s-side or toward the root node on the t-side. The final pass through the arc data allows any unlabeled nodes to be assigned to the t-side by means of arcs directed toward the root. Like the other two start procedures, all network arcs have zero initial flow. It should be noted that it is unnecessary to repeat the initial steps until a pass of the arc data fails to add another arc to the tree (STEP 1C).

The final start procedure tested, called the *modified balanced tree start*, begins the final pass (a modified STEP 1F) as soon as an initial pass (STEPS 1B-1E) adds fewer than n_1 nodes to the basis tree, or as soon as a total of n_2 nodes have been added to the basis tree.

4.3 Pivot Strategies

One of the most crucial aspects of any primal simplex network algorithm is the pivot strategy that is employed to select a non-basic arc to enter the basis. This corresponds to STEP 3 of the primal simplex maximum flow network algorithm. Many different pivot strategies were developed and tested during this study. This includes simple modifications of the pivot strategies that have proven successful for more general network flow problems [20, 21, 36], as well as new strategies developed from insights into the special structure of the maximum flow

network problem. The level of complexity of these pivot strategies range from the simple sequential examination of forward stars to a complex candidate list with a steepest descent evaluation criteria. A brief description of each of the fundamental pivot strategies is presented.

In order for an arc to be pivot eligible its nodes must be on opposite sides of the basis tree. In addition, its flow must be at the appropriate bound. Specifically, arc k = (i,j) is a candidate to enter the basis if and only if

$$\text{(a)} \quad \pi_i = 1,\ \pi_j = 0, \text{ and } x_k = 0 \text{ or}$$

$$\text{(b)} \quad \pi_i = 0,\ \pi_j = 1, \text{ and } x_k = u_k.$$

In case (a), arc k currently has no flow and is directed away from the s-side and toward the t-side. It is advantageous to attempt to increase flow on this arc by pivoting it into the basis. In case (b), arc k currently has as much flow as it can handle and it is directed away from the t-side and toward the s-side. In this instance, it appears to be advantageous to decrease flow on the arc, thus leading to a net increase in flow from the source to the terminal. For example, in Figure 4 arc (4,5) is eligible to enter the basis from its upper bound (case (b) pivot).

In a sense, the occurrance of a case (b) pivot implies that the algorithm previously made a "mistake" by putting too much flow on arc k, since at this point it appears beneficial to decrease flow on the arc. A statistical analysis of the test problems used for this study

indicates that the primal simplex maximum flow network algorithm tends to concentrate on case (a) pivots. For most of the variants of the basic algorithm, over 98% of the pivots were of the case (a) type. This observation motivated the development of some specialized pivot strategies that initially concentrate on the case (a) entering arcs. This yields a two-phase (suboptimization) solution approach. During Phase I, only the case (a) arcs are allowed to enter the basis, and during Phase II, any pivot eligible arc is allowed to enter the basis.

The first class of pivot strategies is called *sequential* because the arcs are examined sequentially. The simplest sequential pivot strategy selects the first pivot eligible arc encountered to enter the basis. A two-phase sequential pivot strategy restricts its initial pivots to the case (a) type. The extent to which this restriction is made can have an impact on the overall solution efficiency of the algorithm. At one extreme, all case (b) pivots are postponed until the very end of the solution process. That is, the algorithm suboptimizes the problem by just allowing the case (a) pivots, then optimizes the problem by allowing both case (a) and case (b) pivots. Another implementation of the two-phase sequential pivot strategy restricts pivots to the case (a) type for the first p_1 pivots (or the first p_2 passes through the arc data).

The three sequential approaches are called, respectively, (1) SEQ/NS (sequential with no suboptimization), (2) SEQ/CS (sequential with complete suboptimization), and (3) SEQ/PS (sequential with partial suboptimization). Clearly, SEQ/PS is the most general sequential pivot

strategy since setting $p_1 = 0(\infty)$ yields the SEQ/NS (SEQ/CS) pivot strategy.

The second group of pivot strategies is called *candidate list* strategies because they involve the use of a list of arcs that are potential candidates to enter the basis [31]. These strategies operate by restricting the choice for the entering arc to arcs contained in the candidate list. Periodically, this list must be reloaded with a fresh set of candidate arcs. The frequency of reloading the list, as well as the length of the list, affect the solution efficiency of the algorithm. Various criteria were studied to control the "quality" of the candidate list. This quality is governed by both the choice of the arcs to place in the list and the choice of the candidate arc to pivot into the basis.

Three criteria were considered in determining the best approach for (re)loading the candidate list. The first criterion is sequential. That is, arcs are loaded into the candidate list by sequentially examining the arc data. The specific implementations of the sequential criterion are labeled SEQ/NS, SEQ/CS, or SEQ/PS, depending upon whether no, complete, or partial suboptimization is desired.

The second criterion for selecting arcs to be placed in the candidate list was motivated by the fact that the amount of difficulty involved in carrying out the list updating procedure (STEPS 2, 5, and 6) depends to a large extent on the number of arcs in the unique (basis equivalent) path between the two nodes of the entering arc, which is the path on which the flow will be changed from the source to the

terminal. Since the basis equivalent path of any pivot eligible arc contains the root, the number of arcs in this path is simply the sum of the depth function values for the two endpoints of the incoming arc.

For these reasons, the second criterion used to reload the candidate list restricts the selection of candidates to arcs whose basis equivalent path contains fewer than p arcs. The choice of the cut-off value, p, is dynamic in nature. Initially p is selected to be small, but to guarantee optimality, p is eventually increased to $|N|$. The implementations of this criterion are labeled BEP/NS, BEP/CS, and BEP/PS, depending upon the level of suboptimization desired.

Computationally, the BEP criteria are more difficult to implement than the SEQ criteria. However, the depth function enables the number of arcs on the basis equivalent path of a pivot eligible arc to be determined quite readily.

The third criterion considered for selecting arcs to be placed in the candidate list is a form of the *steepest descent* criterion. Only arcs that cause a major change in the objective function are considered as candidates. Since the objective of the maximum flow network problem is simply to maximize the flow from the source to the terminal, this criterion reduces to a *largest augmentation* criterion. Each arc placed on the candidate list must allow a flow change of at least q units, where the cut-off value q is selected dynamically. The three implementations of the largest augmentation criterion are called AUG/NS, AUG/CS, and AUG/PS. Because the determination of the allowable flow change associated with a pivot eligible arc requires the complete

traversal of its basis equivalent path, implementations of the AUG criterion are computationally cumbersome.

Slight generalizations of these basic criteria were used to select an entering arc from the candidate list. Depending upon the level of suboptimization, the SEQ criterion simply selects the first pivot eligible arc encountered in the candidate list, the BEP criterion selects the arc in the candidate list with the fewest arcs in its basis equivalent path, and the AUG criterion selects the arc that allows the largest flow augmentation.

Additional considerations for candidate list strategies include rules for controlling the length of the list as well as the frequency of reloading. In general, our tested strategies employ a dynamic length candidate list in which the number of elements is a function of the degree of difficulty involved in locating pivot eligible arcs. Typically, this results in an initial candidate list of twenty to fifty arcs. As optimality is approached, and the identification of pivot eligible arcs becomes harder, the length of the list is reduced to five or fewer arcs. Some of the tested strategies also used a maximum pivot counter to restrict the number of pivots between reloadings.

4.4 Leaving Arc Selection

Only two criteria were considered for determining the leaving arc (STEP 4) from the collection of those that restrict the amount of flow change δ (i.e., that yield the minimum ratio in the standard simplex test for the outgoing variable). The customary choice is simply to

select the first arc that qualifies, and we used this in most of the primal simplex codes developed for this study. However, we also tested a second strategy that has been proposed as a mechanism for controlling cycling. This strategy [3, 7, 8] is simply to select the binding arc closest to the terminal.

4.5 Computational Testing

For the more than twenty variants of the specialized primal simplex maximum flow algorithm developed and tested during this study, the principal questions we considered were:

1) What is the best starting basis? (STEP 1)
2) What is the best entering arc selection rule? (STEP 3)
3) What is the best leaving arc selection rule? (STEP 4)
4) What is the best data structure for storing the original problem data?

To some extent, the determination of the answer to one question depends on the answers to the other three questions. For example, when the original problem data is stored in the "random" format, it is virtually impossible to implement any pivot strategy based on processing a node's forward star.

Choice of Leaving Arc

The first question we address concerns the appropriate choice for the leaving arc (STEP 4). As mentioned earlier, two strategies were tested in order to answer this question. The first strategy is known

as the *network augmenting path* (*NAP*) rule. This strategy requires a special type of basis structure. As specialized for the maximum flow problem, every basis with the NAP structure is characterized by having a positive net capacity for all basic arcs on the t-side of the tree. Given an initial starting basis of this form, the NAP rule assures all subsequent bases will have this form. Of the starting procedures developed for this study, only the LIFO and FIFO label-out starts yield an initial basis tree with the necessary NAP structure.

Two implementations of the primal simplex algorithm were developed to test alternatives for selecting the leaving arc. The first code, NAP, uses the network augmenting path rule to select the leaving arc. The second code, NONAP, uses the simple "first minimum found" rule (selecting the first arc that qualifies to leave the basis). Both codes use the same LIFO label-out start procedure, sequential entering arc selection, and forward star data structure.

Table V presents the results of applying these two codes to the test problem data base. Neither code appears to be a definite winner on the random, multi-terminal, and transit grid problems, but the NONAP code outperforms the NAP code on the hard problems. The reason is simple: the special structure of the hard problem causes virtually every arc on the basis equivalent path to be binding. The NAP code selects the outgoing arc as close as possible to the terminal node, whereas the NONAP code tends to select the outgoing arc as close as possible to the entering arc. In fact, in about 90% of the nondegenerate pivots, the NONAP code selected the outgoing arc to be the same as the

TABLE V

SOLUTION TIMES IN CPU SECONDS

PROBLEM	NAP	NONAP	LIFO	MODBAL	SEQCS	RANDOM
R1	.09 (.12)	.10 (.13)	.09 (.12)	.10 (.13)	.08 (.11)	.11 (.12)
R2	.23 (.28)	.23 (.28)	.19 (.24)	.22 (.26)	.18 (.23)	.25 (.26)
R3	.22 (.28)	.23 (.29)	.19 (.25)	.22 (.28)	.16 (.22)	.21 (.22)
R4	.34 (.41)	.36 (.43)	.32 (.39)	.24 (.31)	.16 (.23)	.23 (.24)
R5	.53 (.64)	.47 (.58)	.38 (.49)	.49 (.60)	.33 (.44)	.47 (.49)
R6	.66 (.80)	.66 (.80)	.52 (.66)	.58 (.72)	.52 (.66)	.58 (.60)
R7	.40 (.51)	.42 (.53)	.40 (.51)	.32 (.43)	.27 (.38)	.36 (.38)
R8	.64 (.81)	.64 (.81)	.55 (.72)	.60 (.77)	.48 (.65)	.51 (.54)
R9	1.04 (1.27)	1.00 (1.23)	1.21 (1.44)	1.03 (1.26)	.84 (1.07)	.93 (.96)
R10	.90 (1.07)	.81 (.98)	.73 (.90)	.39 (.56)	.46 (.63)	.51 (.54)
R11	1.36 (1.58)	1.22 (1.44)	.85 (1.07)	.68 (.90)	.69 (.91)	.77 (.81)
R12	1.99 (2.26)	1.81 (2.08)	2.32 (2.59)	1.75 (2.02)	1.45 (1.72)	1.62 (1.67)
MR1	.29 (.32)	.32 (.35)	.28 (.31)	.28 (.31)	.26 (.29)	.28 (.29)
MR2	.90 (.95)	1.01 (1.06)	.77 (.82)	.59 (.64)	.58 (.63)	.77 (.78)
MR3	.68 (.74)	.87 (.93)	.68 (.74)	.61 (.67)	.56 (.62)	.58 (.59)
MR4	.52 (.59)	.60 (.67)	.54 (.61)	.34 (.41)	.28 (.35)	.42 (.43)
MR5	1.25 (1.36)	1.18 (1.29)	1.25 (1.36)	.88 (.99)	.83 (.94)	1.04 (1.06)
MR6	2.22 (2.36)	2.02 (2.16)	2.08 (2.22)	1.34 (1.48)	1.49 (1.63)	1.92 (1.92)
MR7	1.40 (1.51)	1.24 (1.35)	1.00 (1.11)	.71 (.82)	.64 (.75)	.80 (.82)
MR8	1.98 (2.15)	2.12 (2.29)	1.89 (2.06)	1.02 (1.19)	1.00 (1.17)	.95 (.98)
MR9	4.53 (4.76)	4.41 (4.64)	3.48 (3.71)	2.70 (2.93)	2.52 (2.75)	2.90 (2.93)
MR10	1.36 (1.53)	1.09 (1.26)	1.16 (1.33)	.77 (.94)	.70 (.87)	.77 (.80)
MR11	3.26 (3.48)	3.09 (3.31)	2.09 (2.31)	1.64 (1.86)	1.97 (2.19)	2.14 (2.18)
MR12	6.95 (7.23)	7.13 (7.41)	6.11 (6.39)	4.70 (4.98)	5.39 (5.67)	5.82 (5.87)
TG1	.28 (.33)	.26 (.31)	.45 (.50)	.48 (.53)	.21 (.26)	.42 (.43)
TG2	.21 (.25)	.18 (.22)	.52 (.56)	.39 (.43)	.16 (.20)	.38 (.39)
TG3	.60 (.66)	.56 (.62)	1.03 (1.09)	.86 (.92)	.48 (.54)	.75 (.76)
TG4	.52 (.60)	.49 (.57)	.94 (1.02)	.80 (.88)	.41 (.49)	.79 (.80)
TG5	.93 (1.06)	.96 (1.09)	1.87 (2.00)	1.56 (1.69)	.73 (.86)	1.23 (1.25)
TG6	.73 (.88)	.70 (.85)	1.51 (1.66)	1.21 (1.36)	.58 (.73)	1.38 (1.40)
TG7	1.63 (1.81)	1.62 (1.80)	2.74 (2.92)	2.22 (2.40)	.96 (1.14)	1.91 (1.94)
TG8	1.56 (1.72)	1.50 (1.66)	2.50 (2.66)	1.99 (2.15)	1.12 (1.28)	2.02 (2.05)
H1	.07 (.07)	.06 (.06)	.10 (.10)	.11 (.11)	.06 (.06)	.05 (.05)
H2	.57 (.60)	.52 (.55)	.68 (.71)	.65 (.68)	.43 (.46)	.42 (.42)
H3	1.82 (1.88)	1.70 (1.76)	2.40 (2.46)	2.35 (2.41)	1.39 (1.45)	1.35 (1.36)
H4	4.22 (4.31)	3.98 (4.07)	4.35 (4.44)	4.43 (4.52)	3.22 (3.31)	3.09 (3.10)
H5	8.12 (8.25)	7.67 (7.80)	8.91 (9.04)	8.66 (8.79)	6.16 (6.29)	5.92 (5.93)

* This table provides optimization times and total solution times in parentheses (optimization plus initialization) on The University of Texas' CDC 6600. All times are in cpu seconds and reflect the average times for a number of problems of each size.

entering arc. This is a very easy pivot to perform since the structure of the basis tree remains unchanged. On the other hand, the NAP code selected the outgoing arc to be the entering arc on only about 5% of the nondegenerate pivots, resulting in more "hard" pivots than the NONAP code.

The reason that the NAP and NONAP codes performed basically the same on the other problem topologies may be related to the fact that the entering arcs are not as likely to be binding. Indeed, limited sampling of the test problems indicates the entering arc is binding for only 15% to 20% of the nondegenerate pivots for these other topologies.

Since the network augmenting path rule did not improve solution speeds and is not compatible with the balanced tree start procedures, all further reported computational testing is concerned with codes that use the first minimum found rule for STEP 3.

Choice of Start Procedures

Four basic start procedures (STEP 1) were implemented. The first code, referred to as LIFO, uses the last-in first-out or depth first rule to construct the initial basis tree. The second code, FIFO, uses the similar first-in first-out rule. Both of these procedures require the arcs to be processed in forward star sequence. The balanced tree start was implemented in the code BAL and the modified balanced tree start was implemented in MODBAL. The MODBAL implementation uses the parameter settings $n_1 = .10|N|$ and $n_2 = .75|N|$ which cause the final pass to begin as soon as an initial pass fails to add at least 10% of

the nodes to the tree or as soon as the tree contains at least 75% of the network nodes. Limited testing with other parameter settings indicated that these values were quite robust across all problem topologies.

All four implementations used the same entering arc selection rule. Specifically, a candidate list of length twenty was used. The SEQ/CS criteria was used to load the list and the BEP/NS criteria was used to select entering arcs from the list.

The testing indicated that the LIFO and FIFO codes perform approximately the same in terms of solution time. The starting bases generated by the two codes, however, are quite different. As expected, the LIFO code constructs a thin, deep initial basis tree and the FIFO code constructs a fat, shallow initial basis tree. However, the structure of the optimal basis tree does not resemble that of either initial tree.

The principal shortcoming of both codes is that they initially place all nodes except the terminal on the s-side of the tree. This makes the identification of eligible entering arcs somewhat difficult during the initial pivots, as reflected by the fact that the average number of arcs examined per pivot is much higher during the early pivots than during the middle pivots for both the LIFO and FIFO code.

Computational tests comparing the modified balanced tree start MODBAL and the standard balanced tree start BAL indicate MODBAL is clearly superior. This superiority is most pronounced on the multi-terminal random problem class. On selected problems in this class, MODBAL outperformed BAL on all but one problem.

Due to the similarity of the LIFO and FIFO times, and the superior-

ity of MODBAL over BAL times, only the results for LIFO and MODBAL are presented in Table V. These results indicate that the modified balanced tree start is superior to the other start procedures tested, particularly for the problem classes designed to simulate multi-terminal networks (i.e., multi-terminal random and transit grid).

Choice of Entering Arc

Although many codes were developed to test the impact of the entering arc selection rule (STEP 3), only six of the codes will be presented in any detail. All codes use the modified balanced tree start procedure, the first minimum rule for selecting the leaving arc, and the forward star form for storing the original problem data.

The first three codes, SEQNS, SEQCS, and SEQPS, were developed to test the impact of suboptimization on overall solution speeds. SEQNS uses the sequential pivot selection rule with no suboptimization. SEQCS, on the other hand, uses the same sequential entering arc criterion, but requires complete suboptimization of the problem using only case (a) entering arcs before any case (b) entering arcs are allowed. SEQPS uses the sequential criterion with partial suboptimization. Phase I (case (a) pivots only) was terminated after $p = .50|N|$ pivots.

Computational testing indicated that SEQCS is superior to SEQNS on all but the smallest networks. The superiority of SEQCS is particularly evident on the large multi-terminal random problems where SEQNS ran as much as 60% slower than SEQCS.

The performance of SEQPS is harder to evaluate. Neither SEQPS nor

SEQCS dominated the other. However, SEQCS is highly robust, yielding good solution times for all problems, while SEQPS yields solution times whose quality is far more variable. Barring the possibility of finding a value for the p parameter for SEQPS that improves its stability, the SEQCS code is preferable for situations in which robustness is valued.

The next three codes tested use a candidate list of length twenty to control the selection of entering arc. Each employs the same criterion for reloading and redimensioning the list: reloading occurs when all pivot eligible arcs in the list have been used; redimensioning occurs when a complete pass of the arc data fails to yield enough arcs to fill the list (whereupon the dimension of the list is set equal to the number of pivot eligible arcs actually found).

All three codes use the SEQ/CS criterion to load the candidate list. The first code, referred to as CANSEQ, uses the SEQ/NS criterion to select the entering arc from the candidate list, while the second code, CANBEP, uses the BEP/NS criterion, and the third code, CANAUG, uses the AUG/NS criterion.

CANAUG turned out to be a definite loser. The reason for its poor performance is that for each pivot, the basis equivalent path of every arc in the candidate list must be traversed in order to identify the arc with the maximum minimum ratio. CANAUG tends to require fewer pivots than the other codes tested, but its solution times ranged from 50% to 300% slower. Other implementations of the largest augmentation criteria, including candidate list and non-candidate list codes with no and partial suboptimization, performed just as badly.

Table V presents the solution times for SEQCS and CANBEP. In the table, CANBEP is referred to as MODBAL since it is the same code used to test the choice of starting bases. The performance of these two codes is basically the same on the random and multi-terminal random problems, but SEQCS dominates CANBEP on the transit grid and hard problems. SEQCS runs up to twice as fast as CANBEP on the transit grid problems.

A partial explanation of the poor performance of the minimum basis equivalent path length criterion on the transit grid problems is that it generates more pivots, both total and degenerate, than the sequential criterion. For the transit grid problems, the lengths of the flow augmenting paths (non-degenerate basis equivalent paths) tend to be long. By concentrating on the short basis equivalent paths, CANBEP ends up doing more work than the "less intelligent" approach used in SEQCS.

Choice of Data Structure

The last question considered regards the choice of data structure for storing the original problem data. All codes previously discussed made use of the forward star form. The best such code appears to be SEQCS. A number of codes using the random form were also developed. The best of these, simply referred to as RANDOM, uses the modified balanced tree start procedure, the SEQ/NS entering arc selection rule, and the first minimum found leaving arc selection rule. Table V presents the times for the best forward star form code, SEQCS, and the best random form code, RANDOM. The results are fairly clear: SEQCS is faster than RANDOM for almost all problems, except notably the hard

problems. The most pronounced superiority of SEQCS is for the transit grid problems which have a large number of arcs incident on the source and terminal nodes (problem sets TG2, TG4, TG6, and TG8). On these problems SEQCS runs as much as twice as fast as RANDOM.

The overall conclusion from this computational testing of the primal simplex maximum flow codes is that SEQCS is the most consistent winner. To recapitulate, SEQCS uses the modified balanced tree start (with $n_1 = .10|N|$ and $n_2 = .75|N|$), the sequential entering arc selection rule with complete suboptimization, the first minimum rule for selecting the leaving arc, and the forward star form for storing the original problem data.

Comparison with the Classic Label Tree Approach

We also implemented a number of versions of the standard label tree approach of Ford and Fulkerson [15, 16, 17]. Not only did we investigate the various refinements suggested by Edmonds and Karp [12], Fong and Rao [14], and others, but we also examined a number of alternative data structures. The best of our implementations of the classic approach scans the labeled nodes in a FIFO fashion. The determination of the allowable flow augmentation is postponed until after breakthrough occurs. A portion of the labels are maintained after breakthrough. This enables an advanced start to be made in the FIFO node scanning operation. A powerful data structure, referred to as the *fixed mirror form*, is used to store the representation of the network. A complete description of this data structure is provided in our related study [22].

The solution times for the best overall primal code (SEQCS) and label tree code are presented in Table VI. In all except the random class of problems the primal approach dominates the classic approach.

Another important comparison needs to be made between the two approaches. The computer core storage requirements of the primal method are decidedly superior. Since the maximum flow problem typically is solved as a subproblem within a larger master problem, the amount of core required by the solution procedure can be critical.

All of our computer programs for solving the maximum flow problems are roughly the same size (about two hundred statements). Consequently, the best comparison of core requirements is achieved by expressing them in terms of the dimensions (nodes and arcs) of the network. The primal codes use 7 node length and 2 arc length arrays, whereas the label tree codes use 4 node length and 6 arc length arrays. Since it is typical for the number of arcs to greatly exceed the number of nodes, the label tree code consumes almost three times the core of the primal code.

In summary, the primal approach for solving maximum flow problems appears to be superior to the popular classical approach in both of the critical domains of solution speed and core requirement.

TABLE VI

SOLUTION TIMES FOR THE BEST
PRIMAL AND LABEL TREE CODES

PROBLEM	PRIMAL	LABEL TREE
R1	.08 (.11)	.06 (.13)
R2	.18 (.23)	.14 (.24)
R3	.16 (.22)	.28 (.39)
R4	.16 (.23)	.21 (.32)
R5	.33 (.44)	.41 (.55)
R6	.52 (.66)	.40 (.60)
R7	.27 (.38)	DNR
R8	.48 (.65)	DNR
R9	.84 (1.07)	DNR
R10	.46 (.63)	DNR
R11	.69 (.91)	DNR
R12	1.45 (1.72)	DNR
MR1	.26 (.29)	.52 (.55)
MR2	.58 (.63)	2.70 (2.78)
MR3	.56 (.62)	2.41 (2.51)
MR4	.28 (.35)	.87 (.97)
MR5	.83 (.94)	2.65 (2.80)
MR6	1.49 (1.63)	8.30 (8.49)
MR7	.64 (.75)	DNR
MR8	1.00 (1.17)	DNR
MR9	2.52 (2.75)	DNR
MR10	.70 (.87)	DNR
MR11	1.97 (2.19)	DNR
MR12	5.39 (5.67)	DNR
TG1	.21 (.26)	1.02 (1.07)
TG2	.16 (.20)	1.24 (1.29)
TG3	.48 (.54)	3.27 (3.37)
TG4	.41 (.49)	3.03 (3.13)
TG5	.73 (.86)	DNR
TG6	.58 (.73)	DNR
TG7	.96 (1.14)	DNR
TG8	1.12 (1.28)	DNR
H1	.06 (.06)	.20
H2	.43 (.46)	3.73
H3	1.39 (1.45)	DNR
H4	3.22 (3.31)	DNR
H5	6.16 (6.29)	DNR

* This table provides optimization times and total solution times in parentheses (optimization plus initialization) on The University of Texas' CDC 6600. All times are measured in cpu seconds and reflect the average times for a number of problems of each size.

REFERENCES

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, 1975.

2. R. D. Armstrong, D. Klingman, and D. Whitman, "Implementation and Analysis of a Variant of the Dual Method for the Capacitated Transshipment Problem," Research Report CCS 324, Center for Cybernetic Studies, The University of Texas at Austin, 1978.

3. R. Barr, J. Elam, F. Glover, and D. Klingman, "A Network Augmenting Path Basis Algorithm for Transshipment Problems," Research Report CCS 272, Center for Cybernetic Studies, The University of Texas at Austin. To appear in *An International Symposium Volume on Extremal Methods and Systems Analysis*.

4. R. Barr, F. Glover, and D. Klingman, "Enhancements of Spanning Tree Labeling Procedures for Network Optimization," *INFOR*, 17, 1 (1979) 16-34.

5. G. Bayer, "MAXFLOW, ACM Algorithm 324," *Communications of the ACM*, 11 (1968) 117.

6. T. Cheung, "Computational Comparison of Eight Methods for the Maximum Network Flow Problem," Technical Report 78-07, Department of Computer Sciences, University of Ottawa, Ontario, 1978.

7. W. H. Cunningham, "A Network Simplex Method," *Mathematical Programming*, 11 (1976) 105-116.

8. W. H. Cunningham, "Theoretical Properties of the Network Simplex Method," *Mathematics of Operations Research*, 4 (1979) 196-208.

9. G. B. Dantzig and D. R. Fulkerson, "On the Max-Flow Min-Cut Theorem of Networks," *Annals of Mathematical Studies*, Princeton University Press, Princeton, N.J. (1956) 215-221.

10. R. Dial, F. Glover, D. Karney, and D. klingman, "A Computational Analysis of Alternative Algorithms and Labeling Techniques for Finding Shortest Path Trees," Research Report CCS 291, Center for Cybernetic Studies, The University of Texas at Austin. To appear in *Networks*.

11. E. A. Dinic, "Algorithm for Solution of a Problem of Maximum Flow in a Network with Power Estimation." *Soviet Math. Doklady*, 11 (1970) 1277-1280.

12. J. Edmonds and R. M. Karp, "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems," *Journal of the Association for Computing Machinery*, 19 (1972) 248-264.

13. S. Even and R. E. Tarjan, "Network Flow and Testing Graph Connectivity," *SIAM Journal of Computing*, 4 (1975) 507-518.

14. C. O. Fong and M. R. Rao, "Accelerated Labeling Algorithms for the Maximal Flow Problem with Applications to Transportation and Assignment Problems," Working Paper 7222, Graduate School of Business, University of Rochester (1974).

15. L. R. Ford and D. R. Fulkerson, "Maximal Flow Through a Network," *Canadian Journal of Mathematics*, 8 (1956) 399-404.

16. L. R. Ford and D. R. Fulkerson, "A Simple Algorithm for Finding Maximal Network Flows and an Application to the Hitchcock Problem," *Canadian Journal of Mathematics*, 9 (1957) 210-218.

17. L. R. Ford and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, N.J. (1962).

18. D. R. Fulkerson and G. B. Dantzig, "Computations of Maximal Flows in Networks," *Naval Research Logistics Quarterly*, 2 (1955) 277-283.

19. J. Gilsinn and C. Witzgall, "A Performance Comparison of Labeling Algorithms for Calculating Shortest Path Trees," NBS Technical Note 772, U.S. Department of Commerce (1973).

20. F. Glover, D. Karney, and D. Klingman, "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," *Management Science*, 20 (1974) 793-813.

21. F. Glover, D. Karney, and D. Klingman, "Implementation and Computational Comparisons of Primal, Dual, and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems," *Networks*, 4 (1974) 191-212.

22. F. Glover, D. Klingman, J. Mote, and D. Whitman, "Comprehensive Computer Evaluation and Enhancement of Maximum Flow Algorithms," Research Report CCS 356, Center for Cybernetic Studies, The University of Texas at Austin, 1979.

23. D. Goldfarb and M. D. Grigoriadis, "An Efficient Steepest-Edge Algorithm for Maximum Flow Problems," Tenth International Symposium on Mathematical Programming, Montreal, 1979.

24. R. Helgason, J. Kennington, and J. Lall, "Primal Simplex Network Codes: State-of-the-Art Implementation Technology," Technical Report IEOR 76014, Department of Industrial Engineering and Operations Research, Southern Methodist University (1976).

25. E. L. Johnson, "Networks and Basic Solutions," *Operations Research*, 14 (1966) 619-623.

26. A. V. Karzanov, "Determining the Maximal Flow in a Network by the Method of Preflows," *Soviet Math. Doklady*, 15 (1972) 434-437.

27. B. Kinariwala and A. G. Rao. "Flow Switching Approach to the Maximum Flow Problem: I." *Journal of the Association for Computing Machinery*, 24 (1977) 630-645.

28. D. Klingman, J. Mote, and D. Whitman, "Improving Flow Management and Control Via Improving Shortest Path Analysis," Research Report CCS 322, Center for Cybernetic Studies, The University of Texas at Austin, 1978.

29. P. M. Lin and B. J. Leon, "Improving the Efficiency of Labeling Algorithms for Maximum Flow in Networks," *Proceedings IEEE International Symposium on Circuits and Systems*, (1974) 162-166.

30. V. M. Malhutra, M. P. Kumar, and S. N. Maheshwari, "An $O(|V|^3)$ Algorithm for Finding Maximum Flows in Networks," *Information Processing Letters*, 7, 6 (1978) 277-278.

31. J. Mulvey, "Column Weighting Factors and Other Enhancements to the Augmented Threaded Index Method for Network Optimization," to appear in *Mathematical Programming*.

32. A. Nijenhuis and H. S. Wilf, *Combinatorial Algorithms*, Academic Press (1975) 143-151.

33. S. Phillips and M. I. Dessouky, "The Cut Search Algorithm with Arc Capacities and Lower Bounds," *Management Science*, 25 (1979) 396-404.

34. J. F. Shapiro, *Mathematical Programming: Structures and Algorithms*, John Wiley and Sons, New York, 1979.

35. V. Srinivasan and G. L. Thompson, "Accelerated Algorithms for Labeling and Relabeling Trees with Applications to Distribution Problems," *Journal of the Association for Computing Machinery*, 19 (1972) 712-726.

36. V. Srinivasan and G. L. Thompson, "Benefit-Cost Analysis of Coding Techniques for the Primal Transportation Algorithm," *Journal of the Association for Computing Machinery*, 20 (1973) 194-213.

37. N. Zadeh, "Theoretical Efficiency of the Edmonds-Karp Algorithm for Computing Maximal Flows," *Journal of the Association for Computing Machinery*, 19 (1972) 184-192.